Mapping the Railway formalism onto different domains

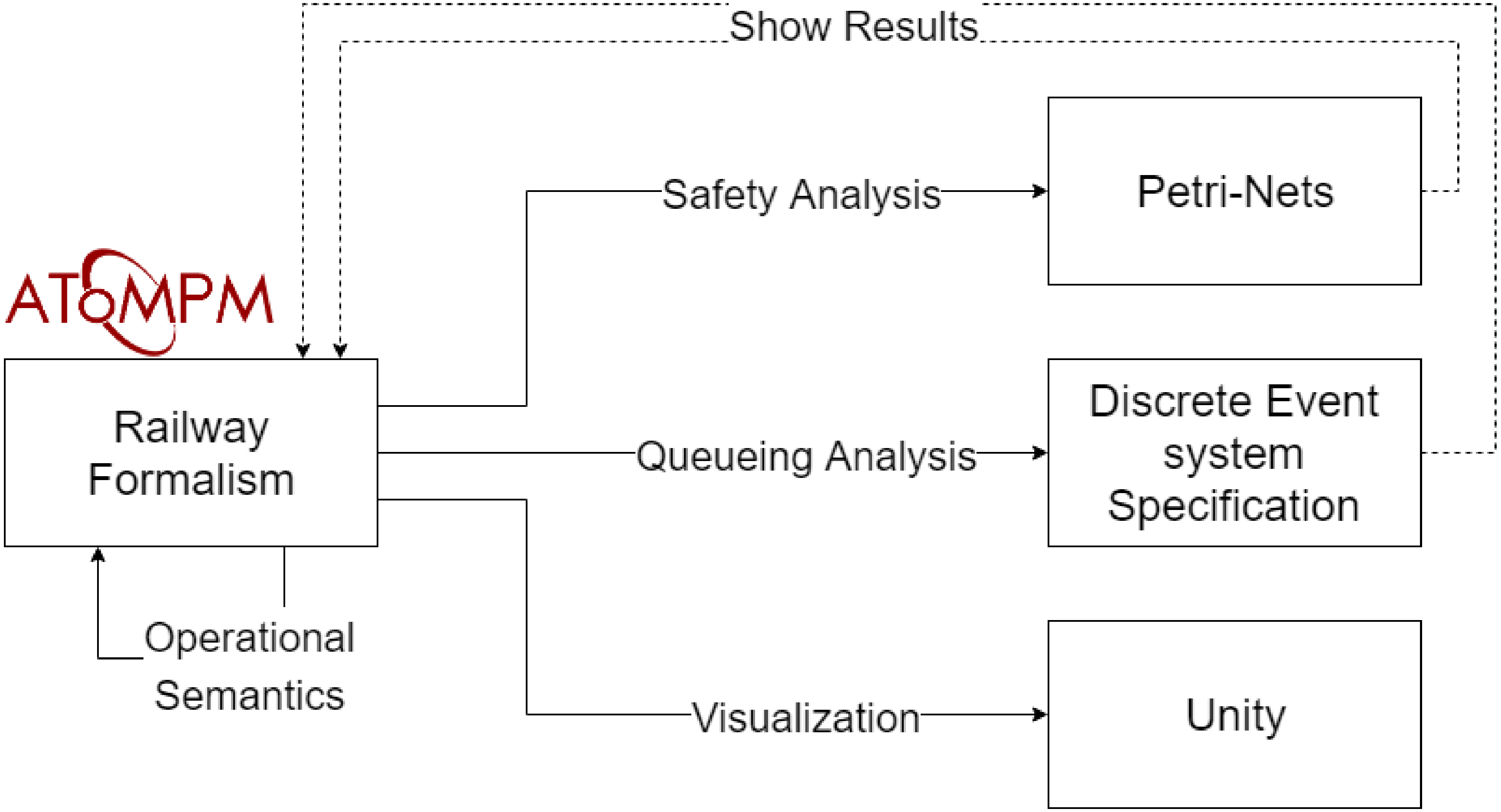
Zhong Xi Lu

Promoter: Hans Vangheluwe

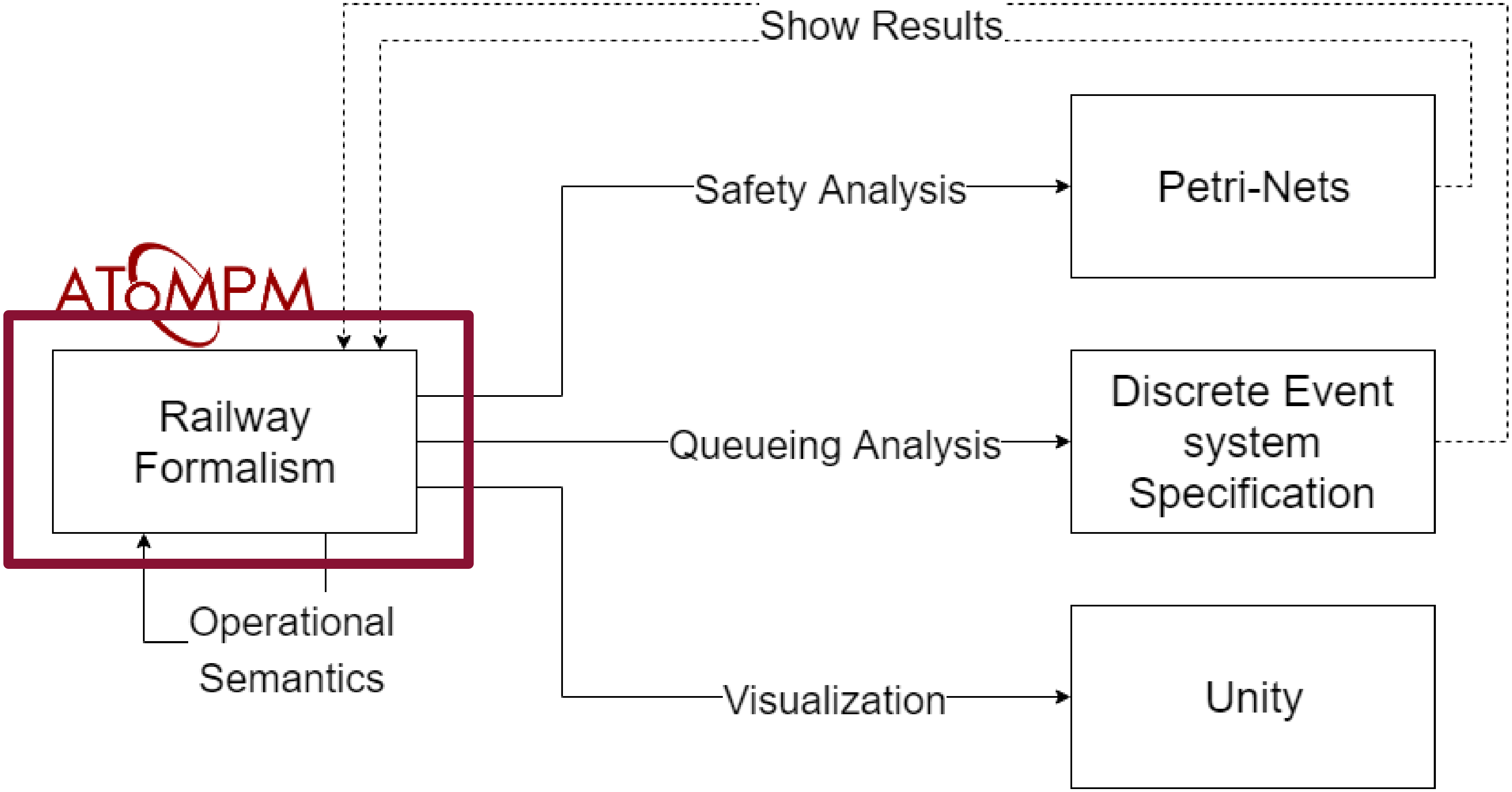
Supervisor: Simon Van Mierlo

University of Antwerp

Overview


Show Results
ATOMPM
Railway Formalism
Safety Analysis
Petri-Nets
Queueing Analysis
Discrete Event system Specification
Visualization
Unity
Operational Semantics

1. Abstract and Concrete Syntax

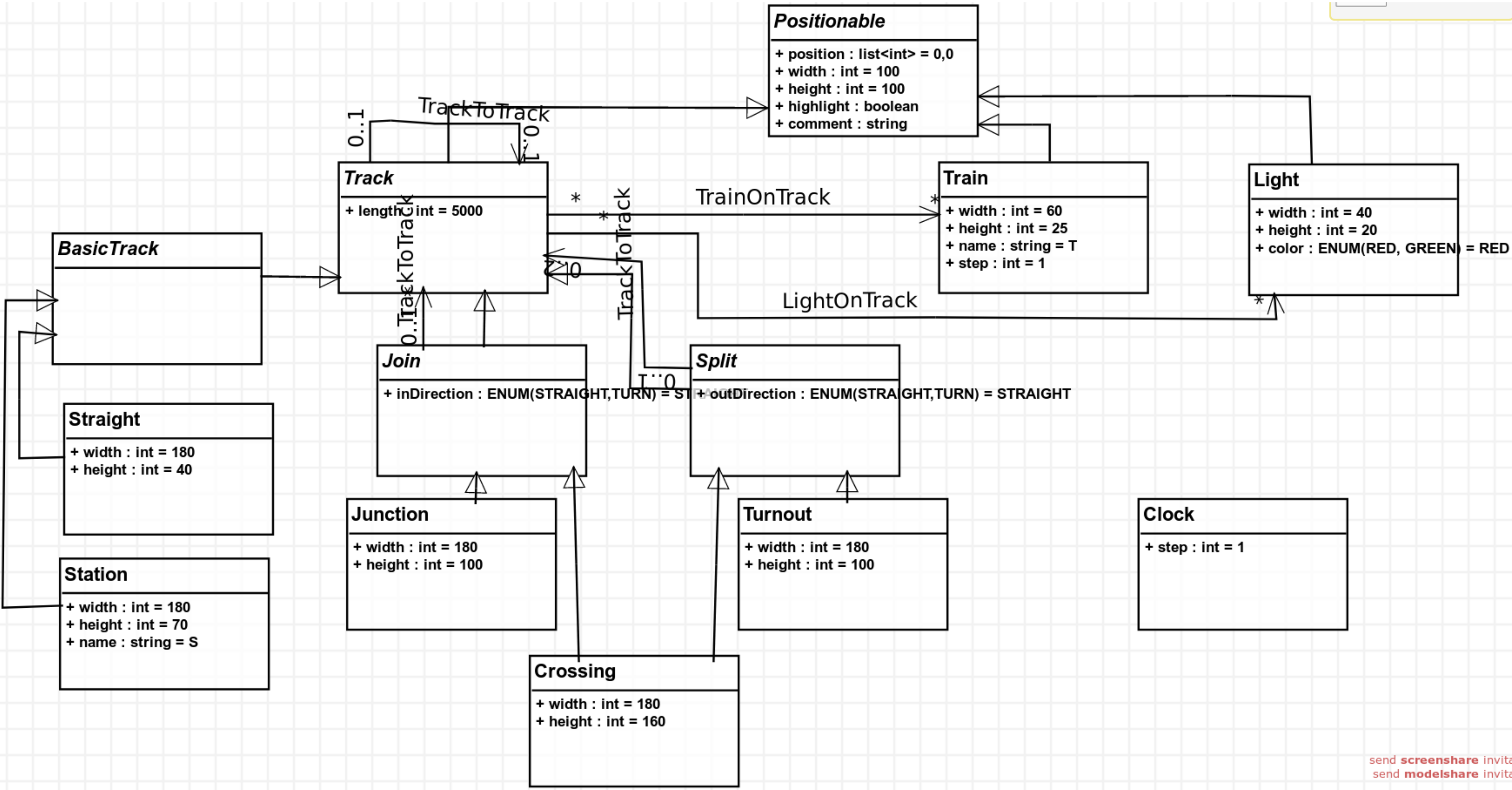


Railway Formalism

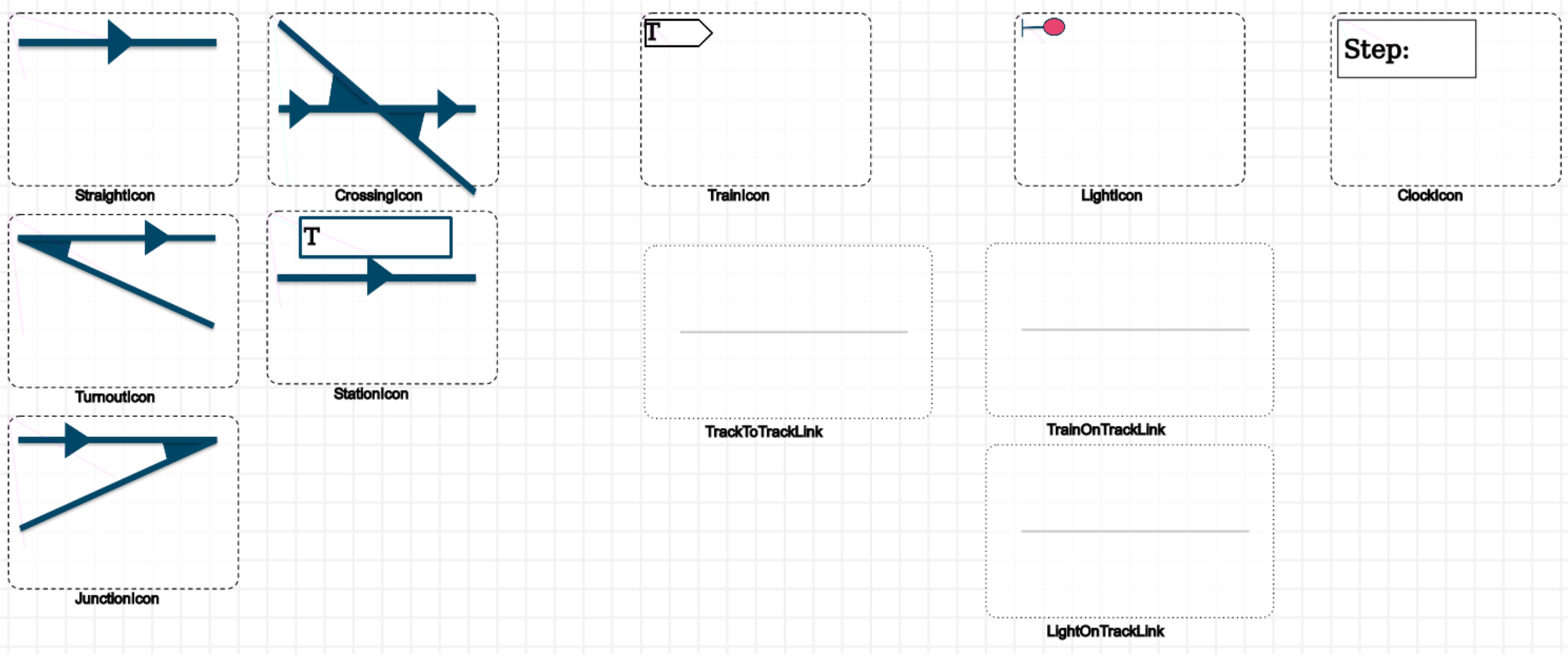
- **Tracks**
  - Straight
  - Turnout
  - Junction
  - Crossing
  - Station
- **Trains**
- **Lights**

Based on: *Railway Operation and Control* by Joern Pachl

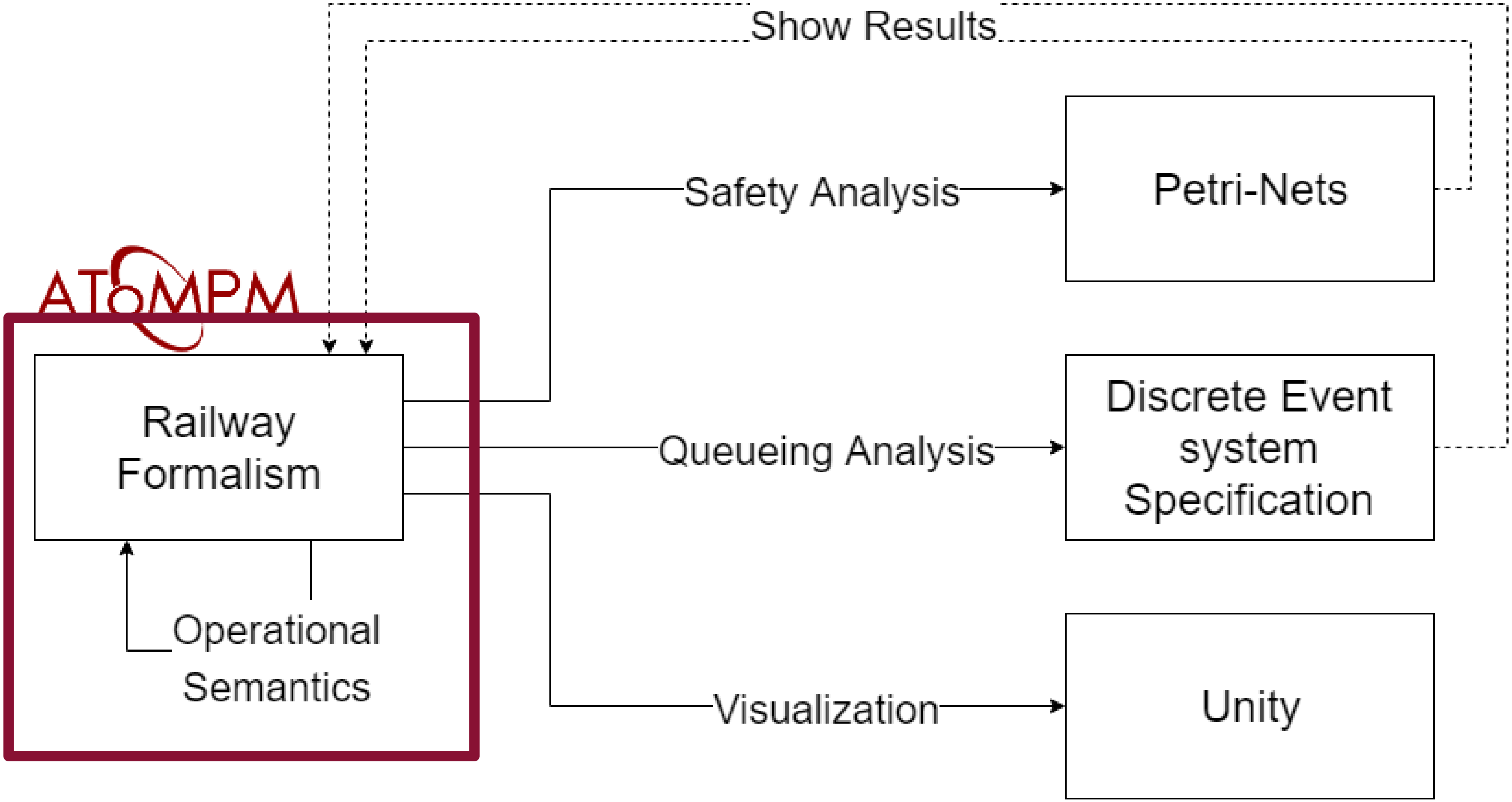
Abstract Syntax



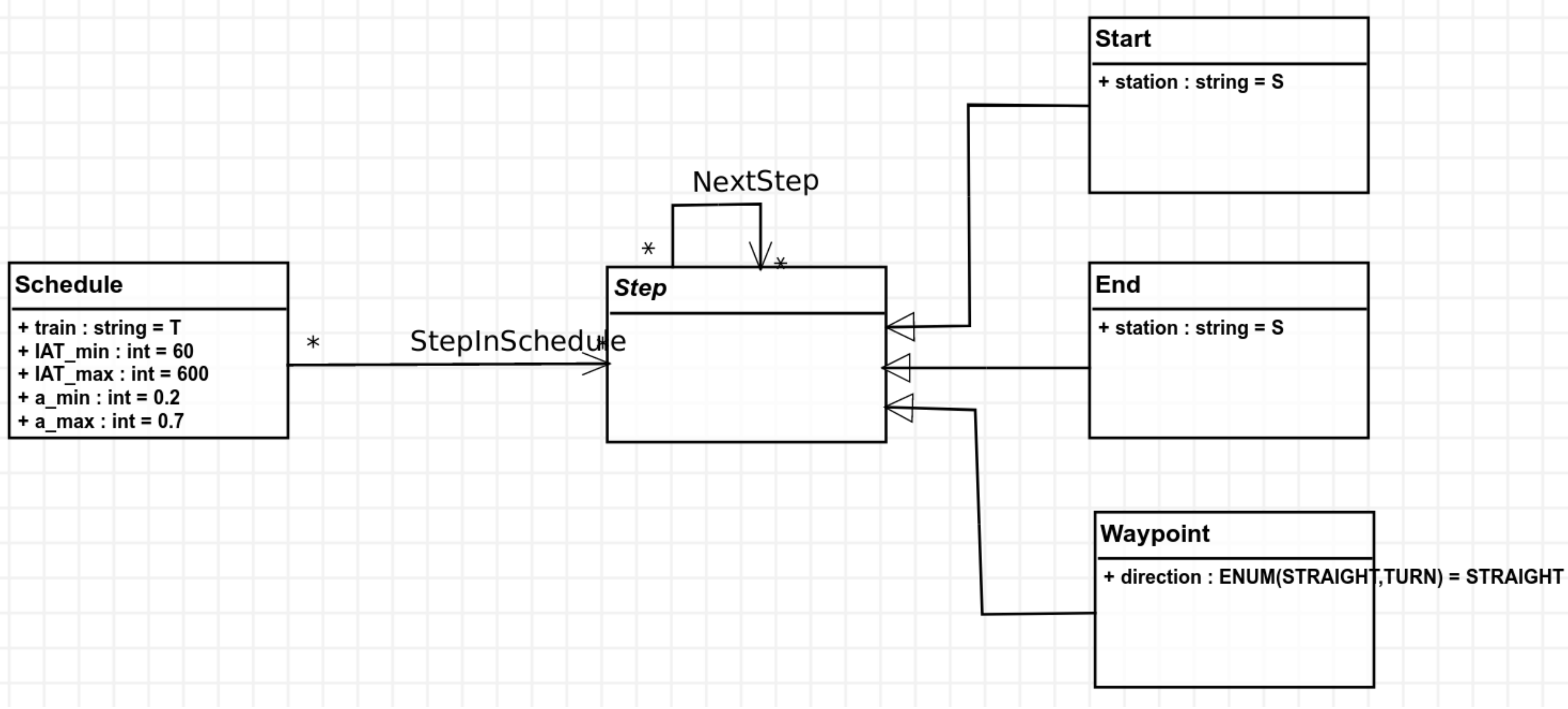
Concrete Syntax



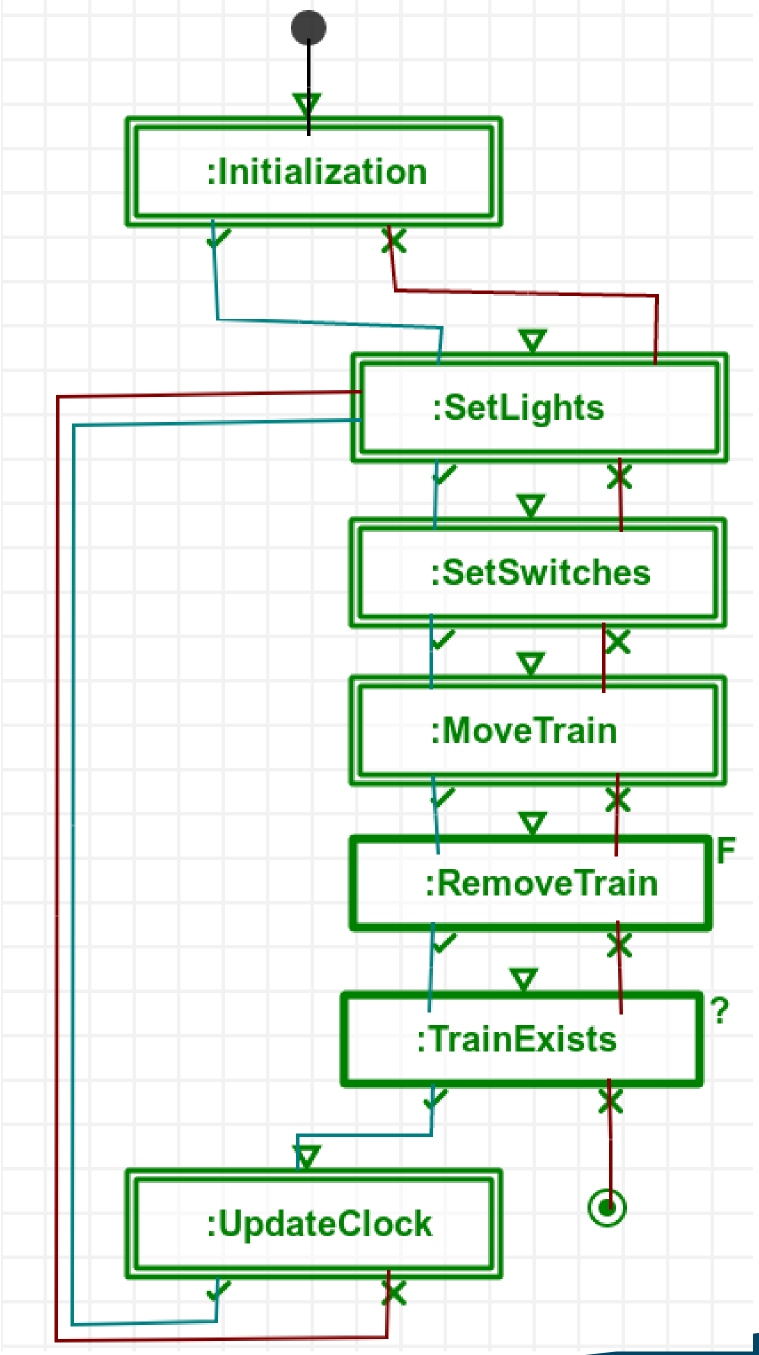
2. Operational Semantics



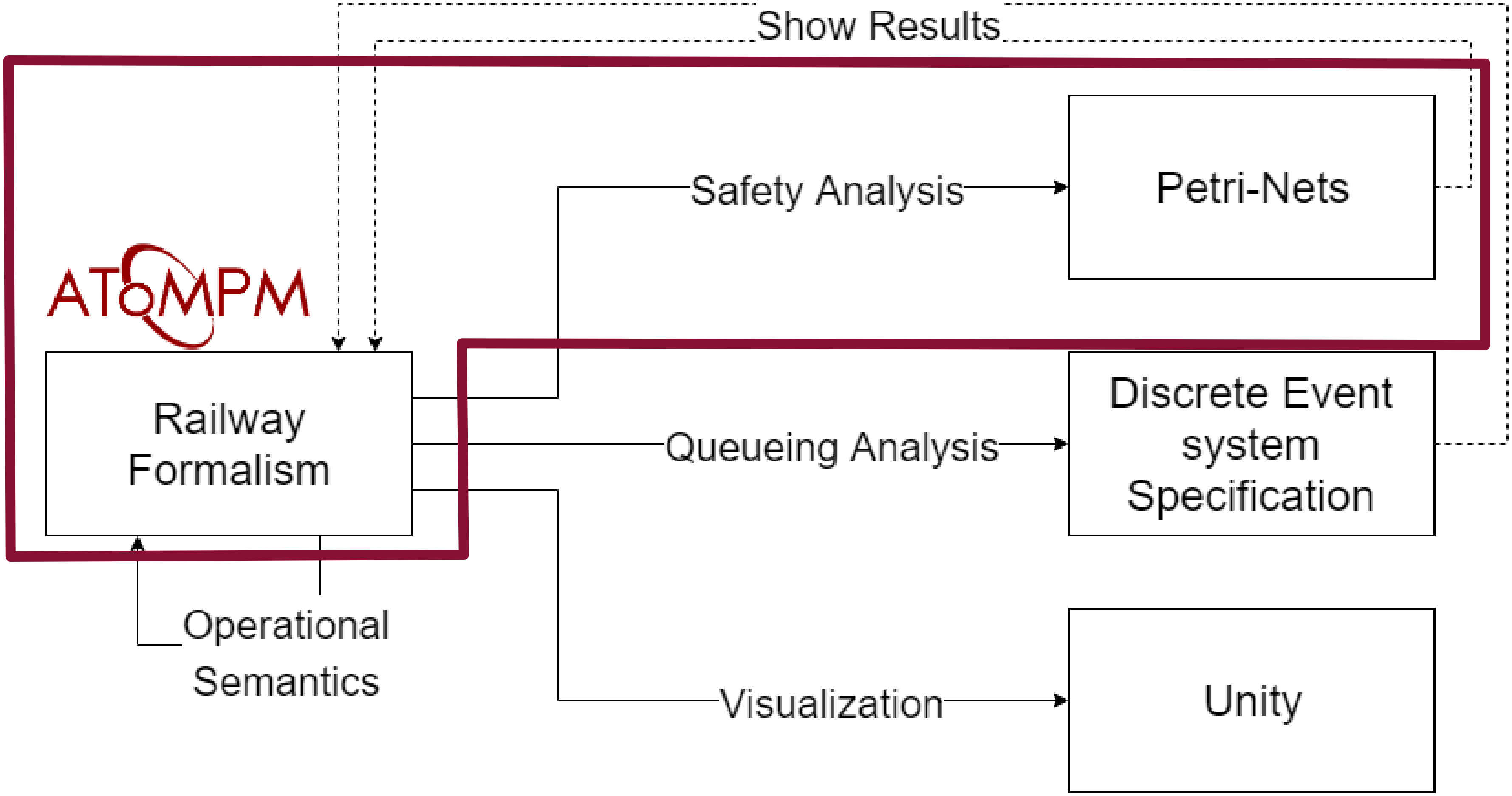
Train Schedule Formalism



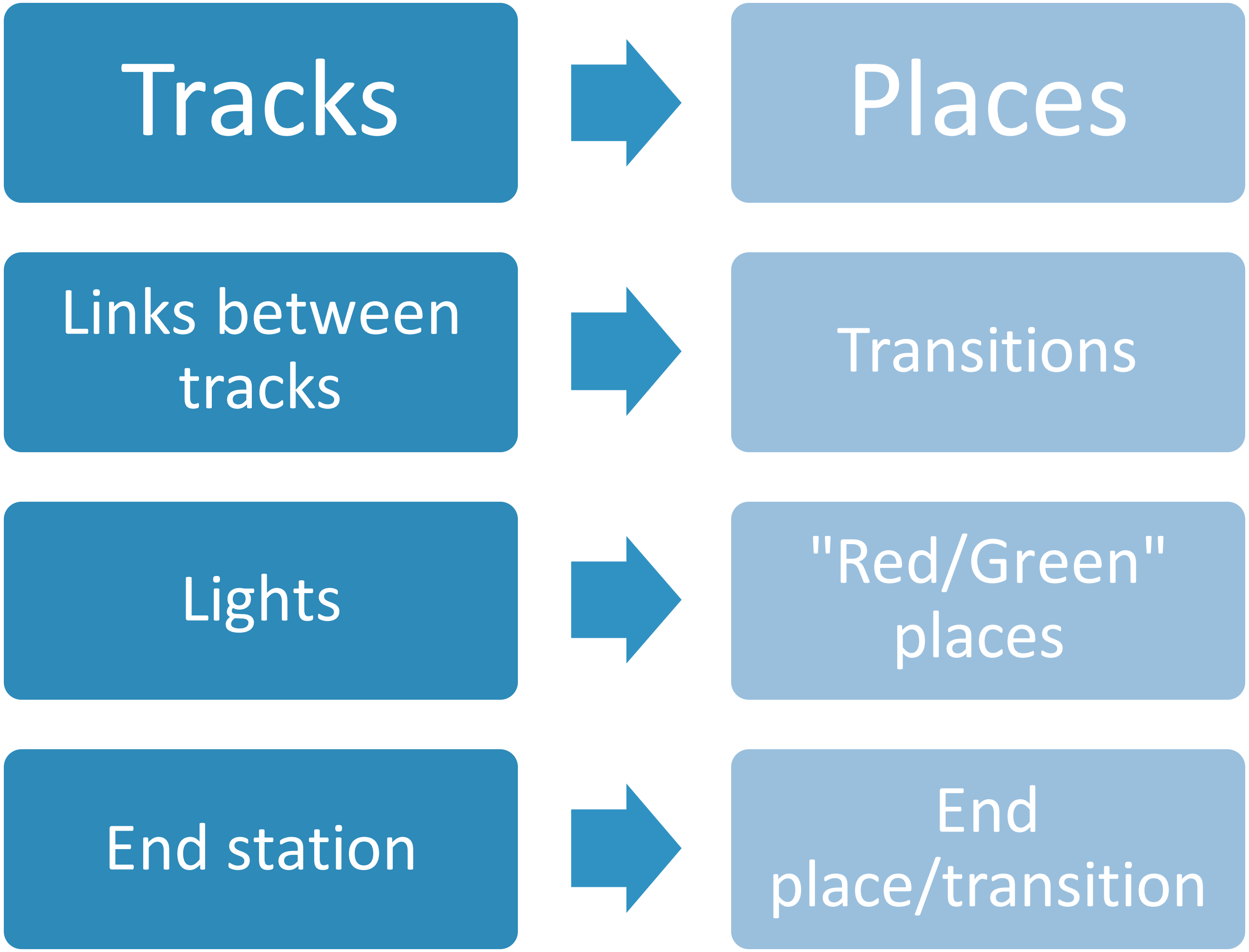
Schedule


:Initialization
:SetLights
:SetSwitches
:MoveTrain
:RemoveTrain
F
:TrainExists
?
:UpdateClock

3. Safety Analysis



Mapping to Petri-nets



Tracks → Places

Links between tracks → Transitions

Lights → "Red/Green" places

End station → End place/transition

LoLA

- A Low Level Petri net Analyzer
- Command line tool
- Specify custom properties through CTL formulas (Computation Tree Logic)

Safety Properties

- Deadlock: EF DEADLOCK
- Reachability: EF $T > 0$
- Safety: AG $T \leq 1$
- Lights Invariant: AG ($G = 1$ OR $R = 1$)

Custom Properties

- New formalism to model properties
- Based on CTL
- Possible to reference particular tracks

Interfaces

- Use ID's for traceability ($atompmId)
- Generate LoLA petri net file
- Call LoLA via command
- Read results from files

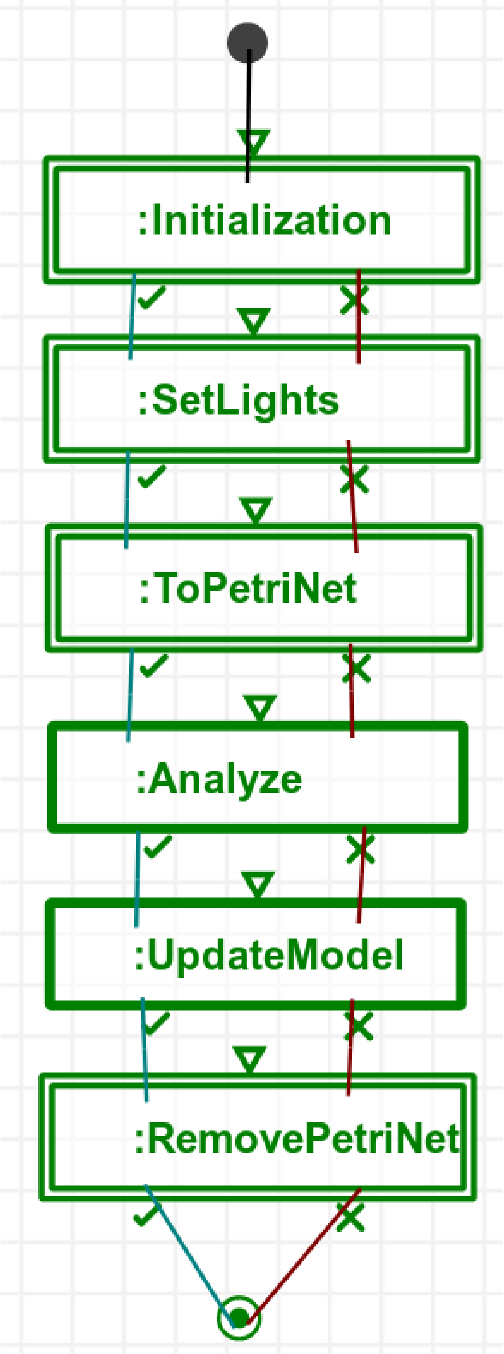
Analyze

UpdateModel

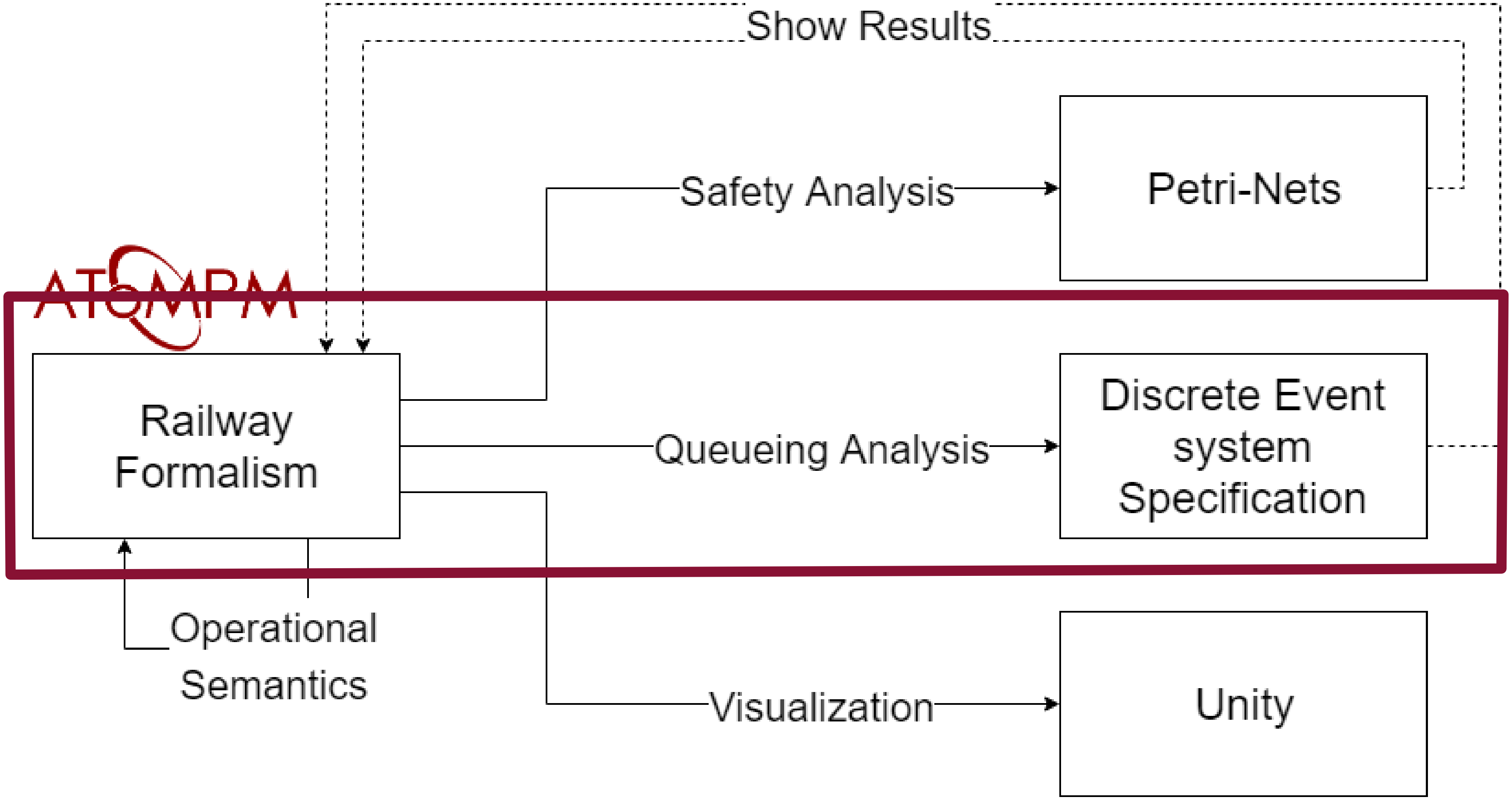
Replay

- Replay trace of counterexample
- Trace generated by LoLA (fired transitions)
- Transformation rules (similar to operational semantics)
    - Based on this trace instead of train schedule

Schedule


:Initialization
:SetLights
:ToPetriNet
:Analyze
:UpdateModel
:RemovePetriNet

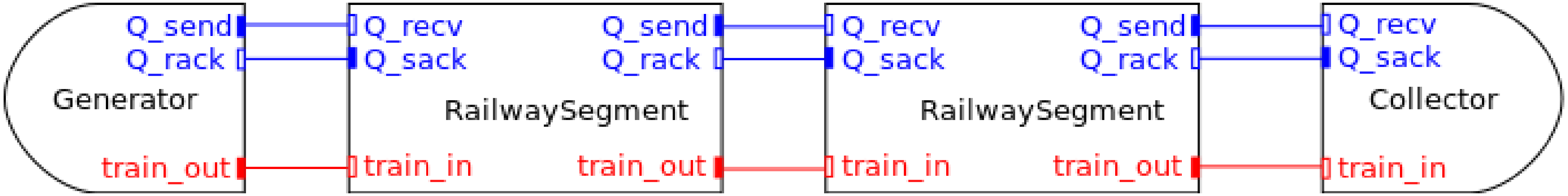
4. Queueing Analysis



DEVS Model

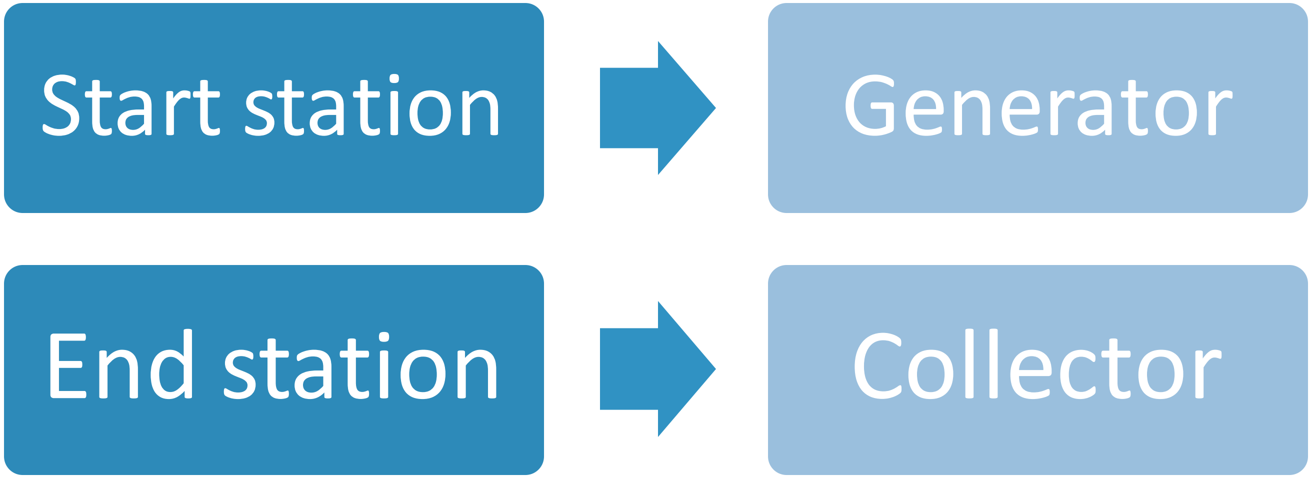
- Using PythonPDEVS
- Atomic models:
  - RailwaySegment
  - Join
  - Split
  - Crossing
  - Generator
  - Collector

DEVS Model Example



Source: http://msdl.cs.mcgill.ca/people/hv/teaching/MoSIS/assignments/DEVS

Mapping to DEVS



Start station → Generator

End station → Collector

Other tracks correspond to atomic DEVS model (e.g. Junction to Join)

Links between tracks become channels (connect ports)

Properties

- Average transit time of schedule
- Throughput of track
- Average transit time of track

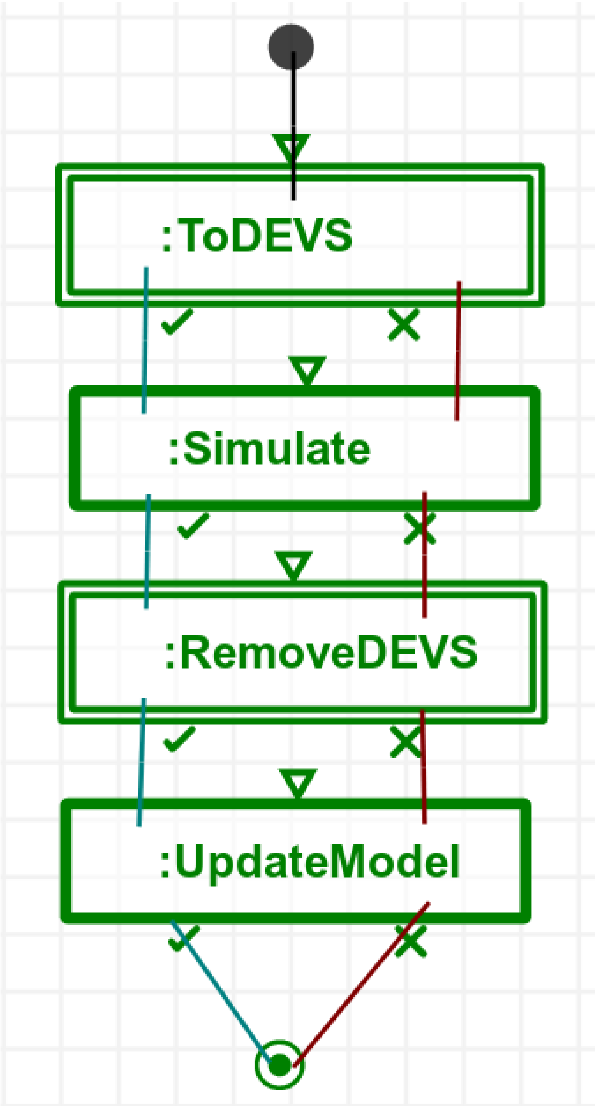
Interfaces

- Use ID's for traceability ($atompmId)
- Generate python PythonPDEVS file
- Call PythonPDEVS to run the simulation
- Read results from file

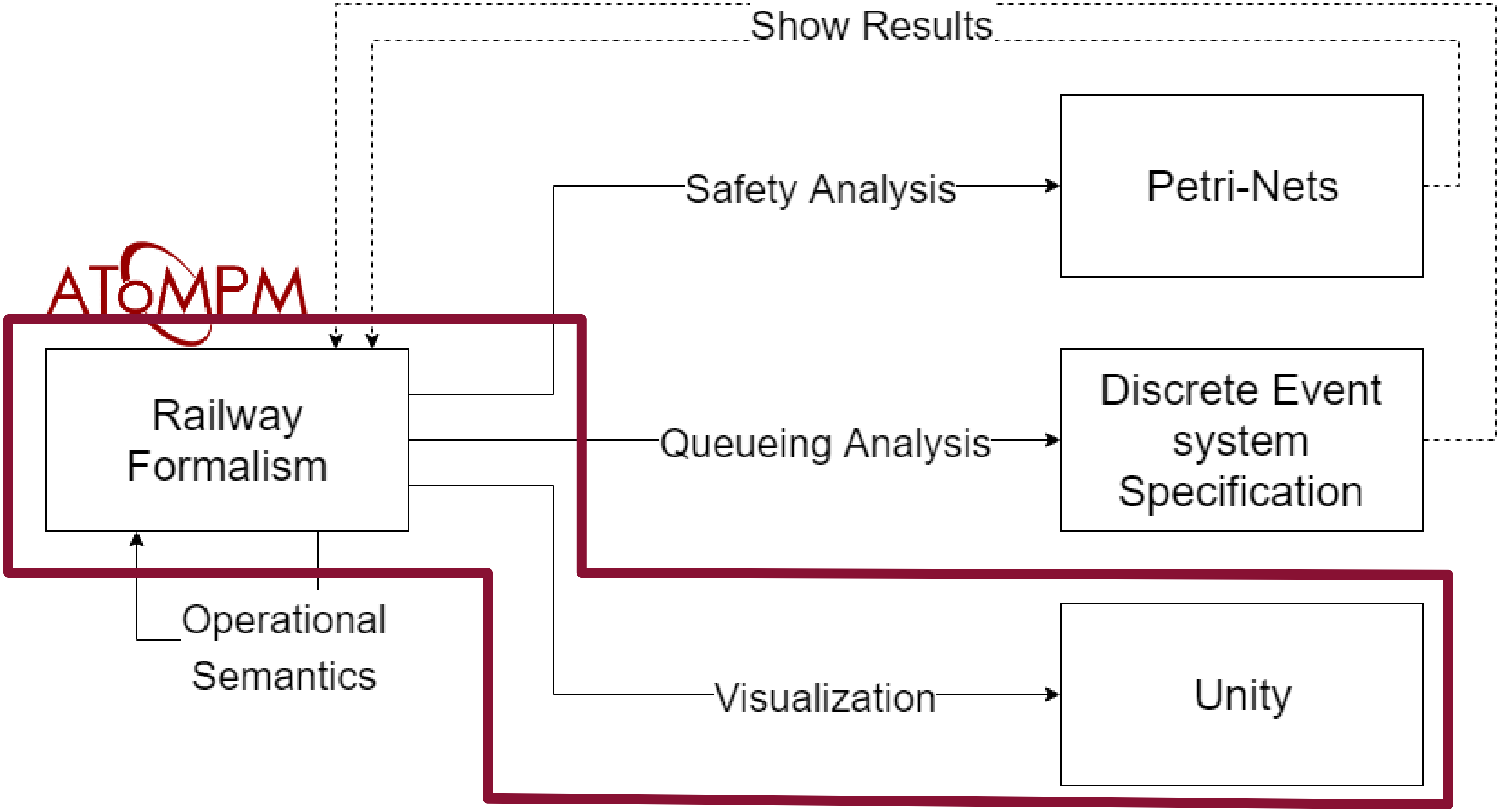
Simulate

UpdateModel

Schedule



5. Visualization



Model Generation

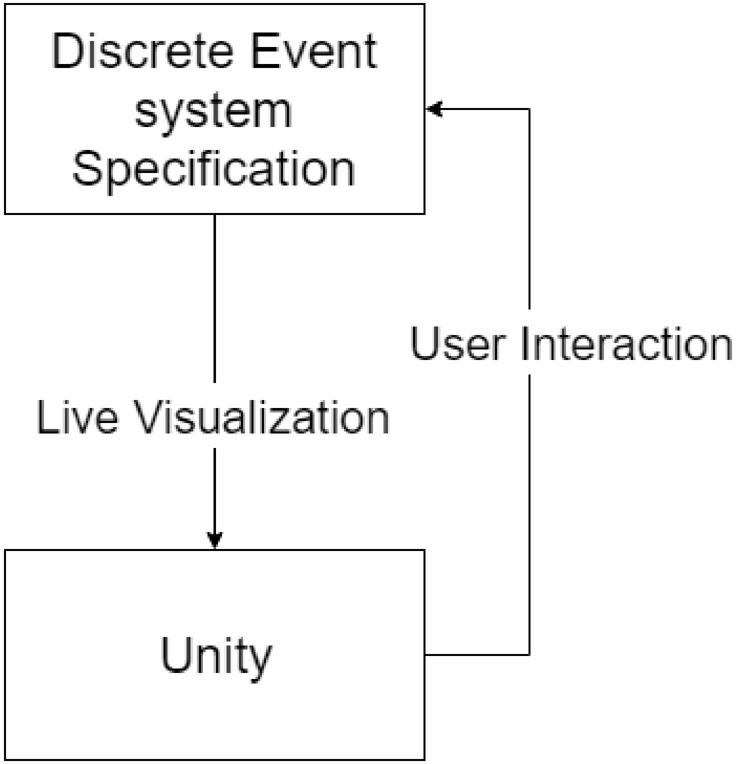
- Using Unity
- Small (xml) file to represent railway network
- Instantiate object in Unity

Simulation

- PythonPDEVS as simulator
- Custom tracer to create tracefile
- Read tracefile to resimulate model
- Gameloop:

```
Update():
    while next event exists:
        if timestamp of next event <= time since startup:
            simulate next event
        else:
            break
```

6. Live Visualization and Interaction



Live Simulation

- PythonPDEVS as simulator
- Custom tracer to ~~create tracefile~~ send messages live to Unity (through sockets)
- Same messages as in tracefile:
  - No "gameloop" anymore

User interaction

- Tweak parameters during simulation
- Send message back to simulator from Unity
- Which model and what parameters to update
- Message interpreted as an external/user event in DEVS